Introducción

Miguel Ángel Solinas 11 mar 2022 (Última modificación: 11 mar 2022)
Javier Jorge 25 mar 2024

Los sistemas compuestos por hardware y software utilizan arquitecturas de capas para desarrollar aplicaciones complejas. En las capas superiores se trabaja con lenguajes de alto nivel, mas "amigables" con el programador. En la capa inferior, más baja, siempre está el hardware puro y duro. Inmediatamente encima está la capa de lenguaje de bajo nivel, podríamos decir más amigable con el hardware.
Recordemos que los lenguajes de bajo nivel están entre uno de los primeros intentos de la humanidad de despegar de la programación directa en lenguaje de máquina. Así el "ensamblador" es un lenguaje propio de la arquitectura y un intento de construir un lenguaje más accesible con el programador.
Los lenguajes de alto nivel, para controlar el hardware y su interacción con los sistemas físicos que lo rodean, necesitan acceder al hardware a través de los lenguajes de bajo nivel. Para ello utilizan convenciones de llamadas.
Entender cómo funciona una convención de llamada nos acercará a un conocimiento de sumo interés para áreas de desarrollo de sistemas críticos, seguridad y también para profundizar sobre el conocimiento de la interacción entre software y hardware.

Bibliografía

El sitio de Paul A. Carter, donde puede descargar los códigos fuentes utilizados en su libro. (http://pacman128.github.io/pcasm/), consultado en Marzo del 2022.-
para compilar primero instalar nasm y librerías de 32bits (*sudo apt-get install nasm gcc-multilib g++-multilib)*

Lenguaje Ensamblador para PC de Paul A. Carter (pcasm-book-spanish.pdf) . Publicado en el 2007, se tradujo a múltiples lenguas y sigue siendo un clásico para comenzar a programar en Ensamblador. Como dice el autor: "El propósito de este libro es dar al lector un mejor entendimiento de cómo trabajan realmente los computadores a un nivel más bajo que los lenguajes de alto nivel como Pascal". Su lectura es mandatoria.

Guía rápida para uso del debugger GDB. Si bien existen otros depuradores con interfaces gráficas mas "friendly", GDB es un clásico y se recomienda fuertemente su utilización.

Enunciado y condiciones de aprobación

Para aprobar el TP#1 se debe diseñar e implementar una interfaz que muestre el índice GINI. La capa superior recuperará la información del banco mundial https://api.worldbank.org/v2/en/country/all/indicator/SI.POV.GINI?format=json&date=2011:2020&per_page=32500&page=1&country=%22Argentina%22. Se recomienda el uso de API Rest y Python. Los datos de consulta realizados deben ser entregados a un programa en C (capa intermedia) que convocará rutinas en ensamblador para que hagan los cálculos de conversión de float a enteros y devuelva el índice de un país como Argentina u otro sumando uno (+1). Luego el programa en C o python mostrará los datos obtenidos.-

Se debe utilizar el stack para convocar, enviar parámetros y devolver resultados. O sea utilizar las convenciones de llamadas de lenguajes de alto nivel a bajo nivel.-

En una primera iteración resolverán todo el trabajo práctico usando c con python sin ensamblador. En la siguiente iteración usarán los conocimientos de ensamblador para completar el tp.

IMPORTANTE: en esta segunda iteración deberán mostrar los resultados con gdb, para ello pueden usar un programa de C puro. Cuando depuren muestran el estado del área de memoria que contiene el stack antes, durante y después de la función.

La defensa del trabajo es GRUPAL (mínimo 2 estudiantes, máximo 3).

Las presentaciones de los trabajos se realizan utilizando GitHub (una cuenta privada). Cada grupo debe asignar un responsable (con email institucional) como usuario de la cuenta de GitHub. Cada usuario tendrá un fork de ese otro usuario. Todos los repositorios estarán sincronizados. Se debe realizar un commit brevemente comentado con cada funcionalidad implementada y validada.- Los usuarios que no sean el representante deberán hacer pull request.

Casos de prueba, diagramas de bloques, diagrama de secuencia, pruebas de performance para comparar c y python son bienvenidos, profiling de la app de c es un plus.

Preparación

En este trabajo haremos una inmersión en los detalles de la interacción entre lenguajes de alto nivel (C) y lenguajes de bajo nivel (Assembler). Si logran usar python es un plus ! Lo importante es que usen C.

Trabajaremos sobre una arquitectura X86 (la que revisamos en el Teórico). Sobre ella tendremos que disponer de un Sistema Operativo Linux.

Ambiente de trabajo:
Linux (Probado con Ubuntu 22.04 64 bits)
Codium, Sublime Text (Instalar Package Control para "view" sintaxis de &) o cualquier otro editor.
Netwide Assembler: Ensamblador libre para X86.
Linkeditor, generalmente viene instalado por default en el SO.
Compilador de lenguaje C, también viene instalado por default
Depurador

sudo apt install build-essential nasm gcc-multilib g++-multilib

NOTA: en gcc, compilar con -g o preferentemente -**g3** para obtener más info de depuración.

En nasm, **-g** también se puede usar -F para elegir el formato de debug. La opción -gformat es igual a -g -F format. Pueden ver los formatos disponibles en nasm -h.

La propuesta es que antes de la primer clase presencial realice las siguientes tareas:

Leer los Capítulos 1 a 4 de la Bibliografía (pcasm-book-spanish.pdf).
Compilar, ejecutar y revisar los programas ejemplos de dichos Capítulos.
Ejecutar y depurar los programas del Cap 4 en el ambiente de trabajo.

Que es una api REST

Una API de REST, o API de RESTful, es una interfaz de programación de aplicaciones (API o API web) que se ajusta a los límites de la arquitectura REST y permite la interacción con los servicios web de RESTful. El informático Roy Fielding es el creador de la transferencia de estado representacional (REST).

Las API son conjuntos de definiciones y protocolos que se utilizan para diseñar e integrar el software de las aplicaciones.Suele considerarse como el contrato entre el proveedor de información y el usuario, donde se establece el contenido que se necesita por parte del consumidor (la llamada) y el que requiere el productor (la respuesta).Por ejemplo, el diseño de una API de servicio meteorológico podría requerir que el usuario escribiera un código postal y que el productor diera una respuesta en dos partes: la primera sería la temperatura máxima y la segunda, la mínima.

En otras palabras, las API le permiten interactuar con una computadora o un sistema para obtener datos o ejecutar una función, de manera que el sistema comprenda la solicitud y la cumpla.

Ejemplo
http://api.open-notify.org/iss-now.json
https://ntfy.sh/
https://realpython.com/api-integration-in-python/

Cómo consumir una api rest en C

la libreria mas popular es libcurl
https://curl.se/libcurl/c/simple.html
sudo apt install libcurl4-openssl-dev

jjorge@pluma:~$ gcc simple.c -lcurl -o simpleget
jjorge@pluma:~$ ./simpleget

Cómo consumir una api rest en python

pip install requests

```python
import requests
res = requests.get('https://scotch.io')
if res:
      print('Response OK')
else:
      print('Response Failed')
print(res.status_code)
print(res)
print(res.text)
```

Como llamar código de c en python

https://es.stackoverflow.com/questions/102982/puedo-insertar-c%C3%B3digo-c-en-python

Tienes muy buenos manuales de ambos métodos en la documentación estándar, para completar la respuesta voy a dejar dos ejemplos simplificados de como usar una función muy simple de C en Python.

> **Nota:** En el ejemplo se va a usar un entorno Linux con gcc 7.2.0 como compilador y Python 3.6.

Imaginemos nuestra función de C para calcular el factorial, algo como:

```cpp
unsigned long long _factorial(int n) {
    unsigned long long result = 1;
    for (int i = 1; i <= n; ++i)
        result *= i;
    return result;
}
```

Bien, vamos a ver cómo extender Python con este código:

Usando ctypes:

1. **Creamos un archivo `factorial.c`** que contiene nuestra función.
2. **Compilamos nuestro código** para crear una librería de enlace dinámico (situándonos en el directorio del archivo fuente):

```
Unset
 $ gcc -shared -W -o libfactorial.so factorial.o
```

Esto nos crea un fichero *shared object* llamado `libfactorial.so`.

3. Ahora que hemos terminado con C vamos a **crear un wrapper con ctypes** para poder usar la librería de C con Python y poder llamar sus funciones:

```python
Python
# Importamos la librería ctypes
import ctypes

# Cargamos la libreria
libfactorial = ctypes.CDLL('./libfactorial.so')

# Definimos los tipos de los argumentos de la función
factorial
libfactorial.factorial.argtypes = (ctypes.c_int,)

# Definimos el tipo del retorno de la función factorial
libfactorial.factorial.restype = ctypes.c_ulonglong

# Creamos nuestra función factorial en Python
# hace de Wrapper para llamar a la función de C
def factorial(num):
    return libfactorial.factorial(num)
```

Existe una incompatibilidad entre libs de 32bits y python 64bits. Esta librería lo soluciona https://msl-loadlib.readthedocs.io/en/stable/index.html esta lib permite en principio usar bins de 32 bits con un intérprete de 64.